APPENDIX B

The Nature of the Problem

MOTIVATION

Optimizing value-added POS transactions for the restaurant industry is a formidably complex task, without even considering the notion of generic business practices. However, suitable AI and machine-learning methods can be implemented which, when presented with sufficient high-quality historical data and clock cycles, will likely be able to outperform hard-coded expert systems by a significant margin. The reason is that the number of optimization parameters is immense, and it would be exceedingly difficult to search the hypothesis space in an efficient manner without utilizing machine learning methods. In addition, the transaction landscape is dynamic with respect to time; optimal strategies continue to change over periods of time, and an ideal optimization logic would satisfy this requirement. In addition, businesses also experience changes in their product line. The maintenance requirements for a diverse set of industries and product inventories is very large. These three factors, dynamic marketplaces, product changes, and maintenance, present a strong motivation to utilize artificial intelligence techniques rather than manual methods.

Reinforcement Learning

Imagine an autonomous agent which is presented with the task of traversing a complex maze repeatedly, seeking one of several exits. Furthermore, imagine that there are different starting points into which the agent is placed. The task of the agent becomes one of learning the maze, and of identifying the minimal distance path to an exit for a random starting location. The agent receives limited information from the environment, such as the shape of the current room, and also is given a restricted set of actions, such as turning left, or moving forwards and backwards.

The task of the autonomous agent falls into the realm of ***reinforcement learning***. Since the agent is not previously presented with optimal solutions nor an evaluation of each action, the agent must repeatedly execute sequences of actions based on states that the agent has

encountered. Furthermore, a reward is distributed at a chosen condition, for example, reaching an exit stage, or after a fixed number of actions have transpired.

Exploitation versus Exploration

The important notions of exploration and exploitation can be evidenced by the example of the *k*-armed bandit problem. An agent is placed in a room with a collection of k gambling machines, a fixed number of pulls, and no deposit required to play each machine. The learning task is to develop an optimal payoff strategy if each gambling machine has a different payoff distribution. Clearly, the agent can choose to pull only a single machine with an above average payoff distribution (reward), but this can still be suboptimal compared to the maximal payoff machine. The agent, therefore, must choose between expending the limited resource, a pull, against a machine with a known payoff (***exploitation***), or instead, to try to learn the payoff distribution of other machines (***exploration***).

The Jupiter Learning Approach

This section serves to present an overview of the methods and logic underlying the Jupiter system, and how Jupiter may be used with embodiments of the present invention.

In any economic exchange, such as a business transaction, there are several parties involved, often the producer or seller, and the consumer. In upsell transactions initiated by a third party, however, the third party itself is another party in the transaction. The fundamental abstract economic principle that guides transaction activity involves a cost-benefit analyses. Summarized, if the benefits of a transaction outweigh the costs, then the transaction is favorable. Furthermore, possible exchanges can be ranked according to this discriminative factor.

In the upsell transaction domain, therefore, there exist three parties, the customer, the host business, and the third party. Jupiter serves as an intelligent broker that seeks to generate upsell offers that are beneficial for *all parties involved.* Consider the consequences of violating this principle. Either the customer would never accept an upsell, the host business would be threatened by "gaming", or the third party would not receive an optimal profit.

Jupiter seeks to create a win-win-win situation for the three parties involved by employing learning technology on two levels. The first level is to determine the maximal utility action that can be performed with respect to the consumer. This is performed by utilizing data mining techniques and unsupervised learning algorithms. Once the possible actions with respect to the consumer have been generated, they are evaluated by a supervised neural network which considers the cost-benefit with respect to the third party and the host business.

The generation of upsell offers can be intrinsically tied in with the consumer needs. However, information should be propagated among any participating establishment, and that any retail sector or business practice is a potential deployment target.

When one asks what knowledge is of the highest utility to be shared in this sort of environment, the answer is the most robust, time-varying, abstract information. In order to achieve the more utility, therefore, knowledge should be represented in as an abstract form as possible. If coincidence dictates that very specific information can be shared, this is also acceptable, but should be considered a by-product of the true utility of the learning/brokering agent.

A sample of such information can be described by the English sentence:
"Offer a high customer benefit item, and also offer an item with high profit to a third party."
One possible GP representation would be:
(SORT OfferRelevancy, SELECT Top, SORT Customer Benefit, SELECT Top)

The Unsupervised Step: Automatically Learning the Domain Using Probabilistic Modeling (Markov Model) and Bayesian Classification

Introduction

Imagine that one is placed in a completely foreign business environment, with the task of fulfilling the upsell generation requirement. An excellent strategy to pursue would be to first observe the transactions that are occurring, and to analyze what items (***resources***) are being sold together. This is because transactions are often initiated in order to satisfy a particular resource need for a customer. In the QSR industry, this may be a food need. In other industries, this may be needs such as children's back-to-school shopping, or a dining room furniture shopping instance.

It would be exceedingly useful if there was a learning method which could:

- Generalize over the items of a transaction
- Produce an upsell tailored for that transaction
- Dynamically and efficiently incorporate new transactions into its learned behavior

This is precisely what the unsupervised learning module of Jupiter seeks to do. The basic idea is that there is a lot of information to be gained from analyses of a particular transaction. This information is amplified through association with a previous memory of past orders over different customers and time frames.

The unsupervised components of Jupiter may utilize both a repository of historical data collected over the entire lifespan of the installation, and in addition, may maintain a "working memory" of the recent transactions that have transpired. This is to account for considerable

deviations from the daily norm which are reflected by processes such as promotions, weather, holidays, and so forth. The weighting of the two distributions can be modified dynamically.

Markov Modeling

A Markov process attempts to describe data using a probabilistic model involving *states* and *transitions*. The idea is that transitions from one state to another are described *probabilistically*, based only on the previous state (the Markov principle). The probability of any arbitrary path through the space of states, therefore, can be assigned a probability based on the transition likelihoods.

In order to account for the inhomogeneities introduced by the termini of sequences, BEGIN and END states are therefore introduced, as illustrated by the graph 900 in FIG. 9:

The Algorithm

A set of nodes, each corresponding to a menu item, are first constructed. The enumeration of the menu items permits the processing of an order as a series of states associated with transitions to states of increasingly greater inventory numeric tags. This therefore disqualifies half of the possible transitions allowed.

A transaction is first converted to a transition path, and the Markov model is modified using these observed values. The probabilities are then renormalized. At this point, the Markov model represents an accurate stochastic description of the transactions that it has observed,as described by the following equation:

$$P(s_b,s_0)P(s_k,s_e)\prod_{i=0}^{k} P(s_i,\ s_{i+1})$$

Offers are generated by calculating the probability of "inserting" an additional transition into the original transaction sequence. All menu items are then potentially assigned a relevancy based on this probability.

Example

A customer places the following transaction:

Items	Jupiter Node Designation
Hamburger	102
Hamburger	102
French Fries	225
Small Coke	332

The transition sequence is then:

(BEGIN, 102), (102, 102), (102, 225), (225, 332), (332, END)

To compute the estimated relevance of an offer, say Apple Pie (node 311), we insert that offer into the transition sequence:

(BEGIN, 102), (102, 102), (102, 225), (225, 311), (311, 332), (332, END)

By multiplying the transition probabilities, we arrive at the total path probability. This is likewise performed for all offers, and these values are then presented to the Jupiter Genetic Programming module along with the Bayes classification (see below).

Markov models are extremely applicable to situations where the state of a system is changing depending on the input (current state). However, they can also be utilized as measures of probability for particular sequences even when the data is derived from a stateless probabilistic process. For example, Markov modeling has successfully been applied to classify regions of genetic information based on the nucleotide sequence. Furthermore, the Markov technique can be used as a *generative* model of the data, in order to derive exemplary paths. The limitation of dependence on the previous state can be overcome by using higher-order or inhomogeneous Markov chains, but the computation becomes much more expensive, and Jupiter presently does not utilize these variants.

Bayesian Classification

The other form of unsupervised, or observation-based learning that Jupiter will employ is a Bayes classifier. The Bayes module will estimate the offer relevancy based on collected data of previous transactions given a set of attributes and values. The set of attributes and values in this case correspond to the internal menu item nodes, with the values being one or zero for inclusion or exclusion in the order.

The target classifications, corresponding to offers, are independent of the orders. This is achieved by only training the Bayes classifier with transactions in which an offer has been accepted. Furthermore, the distribution of the actual order with respect to the offer is irrelevant for training the classifier.

FIG. 10 illustrates in a graph 1000 an example of one menu item node, corresponding to a Coke, representing a target classification. Attributes such as time and general characteristics of the order are included for the classification. The weights extending from the target node correspond to *conditional probabilities* of the target given that particular attribute value.

By calculating the conditional probabilities over the set of attributes and values for each target classification (menu item), the potential offer relevancy (or likelihood of acceptance) can be calculated.

The Learning Algorithm

The Bayes classification module implemented in Jupiter is a variant of a Naïve Bayes Classifier (NBC). The NBC assumes that all attribute values are conditionally independent of each other; this assumption is almost certainly violated in the QSR domain. If the assumption were to hold, then it has been shown that no other learning mechanism using the same prior knowledge and hypothesis space can outperform the NBC. However, in many real-world cases, the independence principle does not hold, but the utility of the NBC is often comparable to the highest-performance algorithms examined.

The Jupiter NBC shall generate estimates for the offer relevancy based on conditional probability over a set of attributes including the time of day, and the inclusion of other menu items in the order. When generating estimates, an m-estimate method shall be utilized which will enable prior knowledge to be integrated into the NBC.

The classifier will then modify the conditional probabilities based on each observed transaction. The task of evaluating a potential offer then becomes one of calculating the conditional probability of the target given the order parameters. In this way, a classification distinct from the Markov approach described earlier is also incorporated into the transaction parameters for evaluation by the genetic programming module (see below).

The Random Model

One of the most important questions one can ask regarding both unsupervised modules described previously and the reinforcement module is the performance versus a completely random approach. Only by comparison of the presently-described learning systems against the random model can an accurate estimation of the utility be derived. Furthermore, this baseline will allow intelligent modifications of the system to achieve better performance. For the prototype, toggles will be present that will allow switching particular modules on/off. For example, bypassing the offer relevancy modules will indicate the magnitude of contribution of the actual order relative to the accept status of the offer in regards to an individual's decision-making process. Factors such as discount percentage might influence the accept decision much more than any other parameters.

The Reinforcement Step: Optimizing the Transaction

Introduction

The reinforcement-learning module is responsible for dealing with the highest level of abstraction, and is entitled with the task of performing the cost-benefit analyses for a transaction. When we considering the notion of exchanging knowledge, this is the primary information that will be exchanged (though as described previously, if knowledge is to be exchanged within the same brand, a larger amount of information can be shared).

The design of the reinforcement learning system consists of evaluating the universal transaction parameters for each party, as illustrated by the diagram 1100 of FIG. 11

As is evident, this type of analyses can be most directly cast as regression analyses utilizing neural networks. In fact, a neural network module has been implemented to achieve this. However, there are several reasons why Genetic Programming (GP) will be utilized instead:

- The evolutionary programming paradigm is more "naturally" amenable to reinforcement learning (e.g.,, an abstract measure of fitness vs. the error surface)
- The situation may be quite dynamic with respect to time; this is further magnified by environments in which multiple Jupiter agents are competing (for example, multiple stores in a local region). This necessitates a learning technique which can react very efficiently to a varying business landscape
- The evolutionary programming paradigm is in the spirit of embodiments of the present invention.
- New terminals, representing additional considerations for the evaluation function for offer inclusion, can easily be inserted.
- The programs can be interpreted and understood by humans more conveniently

There are also advantages to using a neural network representation of the upsell maximization function, but the genetic programming technique will be utilized in the prototype.

The Learning Algorithm

The basic idea behind genetic programming is to evolve both code and data as opposed to data alone. The objective is to create, mutate, mate, and manipulate programs represented as trees in order to search the space of possible solutions to a problem.

As illustrated by the diagram 1200 of FIG. 12, the algorithm consists of generating and maintaining a population of genetic programs represented by sequential programs operating in the Jupiter virtual machine. The programs are then evaluated and assigned a fitness. A new population is then created from the original parental population by selection based on fitness, mating, and mutation. In this manner, solutions to the desired function can be produced efficiently. A population size of 500 was chosen as a starting point for the prototype version based on the estimation that 1000 transactions will be processed per day. This allows every individual to have two opportunities to participate in evaluating an offer. The reason this is important is because since the fitness are distributed according to an absolute measure first (and then normalized), it is very possible for a "good" individual to have been assigned orders that generate a low maximum possible fitness if only one evaluation is performed. Of course, an even greater number of transactions could be processed before generating a new population, but this is a tradeoff between evolution and fitness approximation.

An intriguing possibility is to allow programs to modify themselves during evaluation. This potentially addresses the notion of the Baldwin effect and Lamarckian models of learning and evolution. In molecular biology, there is not necessarily a one to one correlation between the nucleotide sequence and the final protein product; a tremendous amount of regulation and modifications exist in the intermediate stages.

The Jupiter Virtual Machine

Referring to FIG. 13, an embodiment of the Jupiter Virtual Machine 1300 consists of three stacks, a truth bit, an instruction pointer, the instruction list (program), and the input data:

The instruction set for Jupiter Virtual Machine, depicted in TABLES 17 and 18, consists of instructions, which can compare instructions, and transfer or select particular actions.

INSTRUCTIONS	DESCRIPTION
PUSH	Transfer an action from one stack to another.
PUSHT	Transfer an action from one stack to another if the truth bit is on.
PUSHF	Transfer an action from one stack to another if the truth bit is off.
POP	Remove an action from a stack to another.
POPT	Remove an action from a stack to another if the truth bit is on
POPF	Remove an action from a stack to another if the truth bit is off.
>	Compare the top two actions in the operand stack specified by a parameter and set the truth bit if the first has a larger value.
<	Compare the top two actions in the operand stack specified by a parameter and set the truth bit if the first has a smaller value.
EQUALS	Compare the actions specified by two stacks and set the truth bit if they are identical.
SORT	Sort the input stack specified by a parameter into the result stack.
FINDMAX	Find the action with the maximum value for a specified parameter and place into the result stack.

TABLE 17

Jupiter Action Parameters

DISCOUNT PERCENTAGE	BAYES CLASSIFICATION	MARKOV CLASSIFICATION	PROFIT TO THIRD PARTY
PREPARATION TIME	PROMOTION VALUE	INVENTORY	HOST PROFIT

TABLE 18

The above constitute the core instructions utilized in the Jupiter genetic programming module. In addition, architecture-modifying instructions such as automatically defined functions and automatically defined loops allow the generation of more compact and powerful programs. Because each instruction is defined as an object, dynamic generation of new functions is easily accomplished.

The unsupervised modules generate a set of potential offers, each scored separately according to a customer benefit calculation based on the Bayes and Markov activation values. The task of the genetic programs then becomes one of mapping a set of inputs to a set of generated offers.

The separation of abstract pricing information with the semantics of an order constitutes the core of the Jupiter learning system. The system is able to automatically learn the nature of the inventory it is dealing with, but uses abstract ***pricing structure*** information to generate offers. Since the pricing structure information is universal, this knowledge can be shared across any business domain. The pricing structure of an item relates to its discount percentage, promotion value, profit margin, and so forth. This information can apply to any item in any industry. The values are normalized using statistical *z*-scores and relative magnitudes.

The power of evolutionary programming is realized in the potential space that can be searched. However, increasing the size of the space (by the addition of terminals that will not be utilized) can result in a higher amount of computation to achieve a desired level of performance. Therefore, the terminals that have been chosen in Jupiter constitute a basic set of operations rather than an elaborate and exhaustive array of functions.

In addition, if we can *apriori* predict what kind of functions the optimal function will most likely utilize, we can introduce these biases into the genetic programming system as predefined functions. For example, rather than explicitly learning to compute the third-party profit equation, this value is supplied as an input parameter.

FIG. 20 depicts an overview 2000 of one embodiment of the Jupiter Architecture.

Graphical User Interface

The large number of parameters and options available in the Jupiter learning agent necessitates a GUI for monitoring the status of an agent. The GUI allows examination of the

transactions that are pending offer generation, transactions that are pending offer acceptance, and transaction which are pending learning by the Jupiter agent. In addition, visual displays of the Markov model, Bayes classifier, and Genetic programs are accessible to facilitate performance monitoring. An important design issue that had to be considered, however, was the capability to modify the learning parameters. It is unrealistic that anyone outside of the third party (involved in the upsell) would need to do this, or would be sufficiently experienced to do so. Therefore, the ability to change the actual learning process has not been incorporated into the GUI, but can be done outside of the interface.

A description of the primary learning parameters is presented:

- Jupiter Heartbeat
- Unsupervised Module
  - Memory Size
  - *m*-estimate method
- GP
  - Population Size
  - Relative weights for mutation, crossover, architecture-modifications, and Selection

In addition, an evaluation window allows immediate classification by the agent. The GUI is a skeleton model for any Jupiter agent. All that is required is that the agent register with the UI to enable monitoring., using RMI technology. This is illustrated by the diagram 1400 in FIG. 14.

Jupiter Event Model and Control Module

Referring to FIG. 15, the Jupiter agent is composed of a number of different modules, each linked to a state repository and a GUI. Therefore, the propagation of events becomes a crucial issue. This is further compounded by the multi-threaded nature of the Jupiter agent. Therefore, an event model has been developed and implemented that allows changes in component to be detected by other modules which have dependencies on that information. Furthermore, the distributed environment in which multiple Jupiter agents will coexist

simultaneously necessitates a suitable event model 1500 to remotely gather state information pertaining to each agent.

The control module allows dynamic retrieval of the entire menu corresponding to a particular store. The constraints are independent of the industry, and can further be modified online using the GUI. For example, the design enables one to change the price of an item, and then store the modified constraint information back to the database. However, because interoperability issues with other residing systems, such as the POS and DPUM units, this feature has l not yet been. The purpose of the control module is to allow the cost-benefit analyses described previously to occur, independent of the particular store that the agent is in. By either swapping the agent or the control module, knowledge sharing can be implemented.

Validation Filter

The validation filter ensures that only those offers which increase revenue are generated. This is important because the learning methods have some degree of randomness. In addition, the validation filter also ensures that two offers are generated at every instance. In situations where the unsupervised learning may fail to identify two possibilities (with insufficient training), valid offers are created. In situations where the GP module fails to generate the correct number of offers, valid offers are also generated. However, there is no reward received for the action where an item generated by this filter is accepted. Valid offers are probabilistically generated according to pricing and past association. In the absence of a time period designation, and inventory description, these are the two most relevant attributes contributing to offer validity.

The validation filter is not the site at which randomization would be performed to eliminate third party / Customer/Cashier gaming. Rather, it is merely a module, which in at least one embodiment guarantees that the most minimal business requirements are met by guaranteeing offers that never result in a loss, and by guaranteeing that at least two will be presented.

Reward Distributor

The reward distributor is an important modules in the Jupiter system. Because the reinforcement learning is characterized by a mapping from a reward to a fitness, the nature of the reward function guides the evolution of the genetic programs. A GUI may allow the user

to select among a number of possible reward functions, such as accept rates or sales revenue increase.

Transaction Database I/O Interface

The interface supports evaluation of transactions from historical data and from files. In this environment, the optimal performance of the Jupiter agent is defined by the DPUM logic. However, because of the reduced complexity of this environment, because all possible state-action pairs need not be considered, the historical data can serve a useful role as a simulation of an actual commercial environment.

DPUM Integration

The integration with the pre-existing POS/transaction-processing systems may be implemented by using a JNI bridge, or by establishing the Jupiter system as a server proper, and transacting with data over a network connection. The server approach is attractive because it allows the two outside interfaces of a Jupiter agent: with the rest of the Jupiter system, and with the POS array, to be implemented in one module. The JNI approach, on the other hand, is attractive because of the simplicity. In at least one embodiment, the JNI interface is utilized.

Persistent Storage

Persistent storage may be implemented by writing the state of the learning agents into the local database using a JDBC connection. Jupiter may maintain its own set of tables for this purpose. One table may hold the weights for the unsupervised neural network, and an additional table may hold the genetic program population.

Currently, a polling application may draw all the data from a particular store back to a central repository for analyses. This application may be used to also draw all the Jupiter tables back. After analyzing the performance of many stores, appropriate knowledge sharing can be performed.

An exemplary data flow 1600 is illustrated in FIG. 16, which describes both transaction events an the Jupiter Module involved in the event.

Knowledge Sharing

One of the most important theoretical issues regarding capabilities of embodiments of the present invention is the notion of knowledge generalization. We wish to maximize the utility of the system on at least two levels:

- First, the embodiments of the present invention may seek to optimize revenue generated at a particular store, both with respect to the host business, and for a

provider of an embodiment of the present invention. It is therefore important to consider the notion of multi-agent transaction evaluation.

- Second, embodiments of the present invention may seek to distribute knowledge that has been generate from each store, or types of industrial domain, across other business environments.

The knowledge that may be shared includes, for example, the evolved programs. These entities are universal because they operate only in the pricing domain. Each store can then represent a component in the ecosystem, and therefore, each population competes for a niche in the environment.

Knowledge sharing may entail the migration of selected individuals from one store into another.

Agent Architectures

There are several possibilities regarding the architecture of interconnected Jupiter agents.

The ***parallel architecture*** involves a powerful node processing all of the data and generating rewards. The fitness of a large population of genetic programs is evaluated in this manner, and high fitness individuals are then transferred to specific host businesses.

The ***distributed architecture*** involves a single Jupiter agent at each store, with its own population of evolving programs.

Hybrid architectures involve both a central learner (at a third party) in addition to local Jupiter agents. The central learner can generalize across larger regions and has access to a greater number of transactions, whereas the local population can generate programs which are specific to that environment.

Among these, the fully distributed version captures the full power of genetic programming because evolution can occur in parallel among a large number of individuals in different host environments. In the distributed architectures, each store environment can be thought of as a unique ecological niche, and the process of transferring individuals from one population to another can be regarded as a migration process.

Exemplary External Requirements

Processing Requirements
Jupiter may need to be moderately fast CPU at each installation. The actual learning algorithms and classification algorithms may be quite fast (100ms for each transaction), but the procedure of building the unsupervised map may need to be performed over thousands of transactions. This is not required to be performed before each installation, but can be done instead online after the initial install. This is because of the guarantee not to generate inappropriate offers stipulated by the validation filters. Depending on the availability of a historical database, the choice between either online-only or previous batch learning can be made.

An "observation" mode may be employed for Jupiter (e.g., to introduce Jupiter into a completely novel business domain or brand, where the menu would be vastly different from other agents). In such an embodiment, for example, Jupiter may use only its validation filters for a period sufficient to build a representation of the underlying data. This would most likely involve less than a day of observation (depending on the transactional throughput of an installation). The advantages of this approach are:

- Human training or interaction can be obviated
- The learning system can go online within a relatively short period of time
- This enables Jupiter/ embodiments of the invention to more closely resemble an "out-of the-box" solution

Jupiter will not need a central high performance computer. The distributed nature of the system allows the harnessing of hundreds or thousands of CPUs to evolve the population in a distributed fashion. However, the incorporation of Data Warehouse information will not degrade performance, and will permit the generation of more generalized individuals which will augment the locally evolved populations at each installation.

Exemplary Data Requirements
Each Jupiter agent will be instantiated upon startup by the DPUM system. Once the Jupiter agent has been created, flow of information between DPUM and Jupiter may occur via the JNI bridge.

Jupiter may maintain the following persistent storage, as described previously:

- A SQL table corresponding to the weights of the unsupervised network. A rough estimate is that approximately 1-5 M of storage may be required for the network.
- A SQL table corresponding to the individuals in the reinforcement learner population. This is of very variable size, but the estimate is about 500K-1M of storage for the entire population (500 individuals, 1K for each individual)

In addition, Jupiter may also require 2 additional tables for knowledge sharing. One will be utilized by the DPUM polling application in order to store and forward individuals. The other will be a repository for organisms that have migrated into the store.

- A store-and-forward SQL table which contains the individuals that are migrating from one store into another. The maximum size of this table is of course, the maximum size of the population in the store (1M).
- A repository SQL table which contains individuals which have migrated into the target store.

Exemplary Communications Requirements

In the absence of high-speed/continuous links between stores, communication between Jupiter agents may necessitate a central "dispatcher" at a third party which shares agent information. The polling application that draws data from each store can be utilized to achieve this.

The possible of a fast/continuous connection among stores permits the circumvention of this step, and Jupiter agents will be able to directly share information with other, and remote offer generation will be possible.

EXEMPLARY REQUIREMENTS

Within Store (fast, continuous)

- Access to local store's database for storing/retrieving transactions
- Access to local store's database for storing/retrieving state information

Between Store and third party (slow, intermittent)

- Access to Data Warehouse for forwarding state information (knowledge sharing)

OPTIONAL

Between Stores (slow, intermittent)

- Access to other stores' databases for storing/retrieving state information

Between Stores (fast, continuous)

- Remote offer generation
- Access to other stores' databases for storing/retrieving state information

Between Store and third party (fast, intermittent) or (slow, continuous)

- Remote configuration

Between Store and third party (fast, continuous)

- Centralized learning version
- Real-time remote monitoring of Jupiter activity
- Remote configuration

A diagram 1700 of the Jupiter system is illustrated in FIG. 17.

FIG. 18 depicts a window 1800 which describes the Jupiter control module (pricing/inventory information), the unsupervised learner (Resource), and the console for a single-step through a historical transaction. The order is displayed, along with the environment variables, and the classification (after filtering) of the unsupervised learner. The supervised parameters are then evaluated for each unsupervised classification. These will be the parameters that the reinforcement learner will have access to. Not shown in FIG. 18 is the transaction queues, which reveal the transactions waiting for offers to be generated, those that are waiting to be rewarded, and those that are waiting to be learned.

FIG. 19 depicts an evaluation dialog 1900 whereby the user can manually place an order to analyze the system. Menu items can be selected, the quantity specified, and a payment made. After evaluation, a full trace of the transactional through each of the modules is reported, along with the final offers.

Additional features:

- Learning of retail resource associations through unsupervised observation

A crucial feature of Jupiter is its ability to automatically learn the resource distributions and resource associations through observation using unsupervised learning methods. This enables the upsell optimization system to participate in an industrial domain, brand, or store without prior knowledge representation. As transactions are observed, the performance increases correspondingly.

- Genetic programming to enhance upsell performance

The use of genetic programming to automatically create upsell optimization strategies evaluated by business attributes such as profitably and accept rate. Because this is independent of the particular retail sector, this knowledge can be shared universally with other Jupiter agents in other domains.

- Use of a multi-component unsupervised-reinforcement learning system to optimize upsell offers.

Combining unsupervised and reinforcement learning techniques to automatically learn associations between resources, and to automatically generate optimized strategies. This is another key feature of the Jupiter system. By disentangling the resource learning module from the upsell maximizing module, we are able to share the relevant, universal information across any retail outlet. The final feature related to this design is that the reward can be specified dynamically with respect to time, and independently of a domain.

As will be apparent to those of ordinary skill in the art, various embodiments of the present invention can employ many different philosophical and mathematical principals and techniques, such as simple statistical systems and genetic algorithms. Described below are several known methods that could be used to implement embodiments of the present invention.

DATA MINING

Data mining is the search for valuable information in a dataset. Data mining problems fall into the two main categories: classification and estimation. Classification is the process of associating a data example with a class. These classes may be predefined or

discovered during the classification process. Estimation is the generation of a numerical value based on a data example. An example is estimating a person's age based on his physical characteristics. Estimation problems can be thought of as classification problems where there are an infinite number of classes.

Predictive data mining is a search for valuable information in a dataset that can be generalized in such a way to be used to classify or estimate future examples.

The common data mining techniques are clustering, classification rules, decision trees, association rules, regression, neural networks and statistical modeling.

DECISION TREES

Decision trees are a classification technique where nodes in the tree test certain attributes of the data example and the leaves represent the classes. Future data examples can be classified be applying them to the tree.

CLASSIFICATION RULES

Classification rules are an alternative to decision trees. The condition of the rule is similar to the nodes of the tree and represents the attribute tests and the conclusion of the rule represents the class. Both classification rules and decision trees are popular because the models that they produce are easy to understand and implement.

ASSOCIATION RULES

Association Rules are similar to classification rules except that they can be used to predict any attribute not just the class.

STATISTICAL MODELING

A common statistical modeling technique is based on Baye's rule to return the likelihood that an example belongs to a class. Another statistical modeling approach is Bayesian networks. Bayesian networks are graphical representations of complex probability distributions. The nodes in the graph represent random variables, and edges between the

nodes represent logical dependencies. In one embodiment, Baye's Rule may be used to determine that an offer will be accepted given an offer price and the items in the order.

REGRESSION

Regression algorithms are used when the data to be modeled takes on a structure that can be described by a known mathematical expression. Typical regression algorithms are linear and logistic.

CLUSTER ANALYSIS

The aim of cluster analysis is to partition a given set of data into subsets or clusters such that the data within each cluster is as similar as possible. A common clustering algorithm is K Means Clustering. This is used to extract a given number, K, of partitions from the data.

FUZZY CLUSTER ANALYSIS

Like cluster analysis, fuzzy cluster analysis is the search for regular patterns in a dataset. While cluster analysis searches for an unambiguous mapping of data to clusters, fuzzy cluster analysis returns the degrees of membership that specify to what extent the data belongs to the clusters. Common approaches to fuzzy clustering involve the optimization of an objective function. An objective function assigns an error to each possible cluster arrangement based on the distance between the data and the clusters. Other approaches to fuzzy clustering ignore the objective function in favor of a more general approach called Alternating Cluster Estimation. A nice feature of fuzzy cluster analysis is that the computed clusters can be interpreted as human readable if-then rules.

NEURAL NETWORKS ("NEURAL NETS")

Neural nets attempt to mimic and exploit the parallel processing capability of the human brain in order to deal with precisely the kinds of problems that the human brain itself

is well adapted for. Neural networks algorithms fall into two categories: supervised and unsupervised.

The supervised methods are known as Bi-directional Associative Memory (BAM), ADALINE and Backward propagation. These approaches all begin by training the networks with input examples and their desired outputs. Learning occurs by minimizing the errors encountered when sorting the inputs into the desired outputs. After the network has been trained, the network can be used to categorize any new input.

The Kohonen self-organizing neural network (SON) is a method for organizing data into clusters according to the data's inherent relationships. This method is appealing because the underlying clusters do not have to be specified beforehand but are learned via the unsupervised nature of this algorithm.

Exemplary applications to the present invention include, but are not limited to, the following:

- To predict which items are likely to be accepted for a given order.
- To predict the likelihood that a given item will be accepted for a given order.
- To cluster similar orders together
- To classify order items into categories
- To understand how changes in one variable of the data affects another. More specifically, to determine if something like the day of the week or the offer price affects the rate of acceptance. This is called a Sensitivity Analysis.
- Can be used in concert with some of the evolutionary techniques discussed below. For example, the outputted classes or estimations can be used as variables in an evolutionary algorithm.
- The output of many of the algorithms can be translated to human readable rules.

One of ordinary skill in the art may refer to the following references which describe Data Mining:

Fuzzy Cluster Analysis, Methods for Classification, Data Analysis and Image Recognition, Frank Hoppner, Frank Klawonn, Rudolf Kruse, Thomas Runkler, 1999, John Wiley & Sons Ltd

Machine Learning and Data Mining Methods and Applications, Ryszard S. Michalski, Ivan Bratko, Miroslav Kubat, 1998, John Wiley & Sons Ltd

Solving Data Mining Problems Through Pattern Recognition, Ruby L. Kennedy, Yuchun Lee, Benjamin Van Roy, Christopher D. Reed, Richard P. Lippman, 1995-1997, Prentice-Hall, Inc.

Data Mining, Ian H. Witten, Eibe Frank, 2000, Academic Press

Object-Oriented Neural Networks in C++, Joey Rogers, 1997, Academic Press

EVOLUTIONARY ALGORITHMS

Evolutionary Algorithms are generally considered search and optimization methods that include evolution strategies, genetic algorithms, ant algorithms and genetic programming. While data mining is reasoning based on observed cases, evolutionary algorithms use reinforcement learning. Reinforcement learning is an unsupervised learning method that produces candidate solutions via evolution. A good solution receives positive reinforcement and a bad solution receives negative reinforcement. Offers that are accepted by the customer are given positive reinforcement and will be allowed to live. Offers that are not accepted by the customer will not be allowed to live. Over time, the system will evolve a set of offers that are the most likely to be accepted by the customer given a set of circumstances.

GENETIC ALGORITHMS

Genetic Algorithms (GAs) are search algorithms based on the concept of natural selection. The basic idea is to evolve a population of candidate solutions to a given problem by operations that mimic natural selection. Genetic algorithms start with a random population of

solutions. Each solution is evaluated and the best or fittest solutions are selected from the population. The selected solutions undergo the operations of crossover and mutation to create new solutions. These new offspring solutions are inserted into the population for evaluation. It is important to note that GAs do not try all possible solutions to a problem but rather use a directed search to examine a small fraction of the search space.

CLASSIFIER SYSTEMS

One example of a genetic algorithm is a classifier system. A classifier system is a machine learning system that uses "if-then" rules, called classifiers, to react to and learn about its environment. A classifier system has three parts: the performance system, the learning system and the rule discovery system. The performance system is responsible for reacting to the environment. When an input is received from the environment, the performance system searches the population of classifiers for a classifier whose "if" matches the input. When a match is found, the "then" of the matching classifier is returned to the environment. The environment performs the action indicated by the "then" and returns a scalar reward to the classifier system. One should note that the performance system is not adaptive; it just reacts to the environment. It is the job of the learning system to use the reward to reevaluate the usefulness of the matching classifier. Each classifier is assigned a strength that is a measure of how useful the classifier has been in the past. The system learns by modifying the measure of strength for each of its classifiers. When the environment sends a positive reward then the strength of the matching classifier is increased and vice versa. This measure of strength is used for two purposes: when the system is presented with an input that matches more than one classifier in the population, the action of the classifier with the highest strength will be selected. The system has "learned" which classifiers are better. The other use of strength is employed by the classifier system's third part, the rule discovery system. If the system does not try new actions on a regular basis then it will stagnate. The rule discovery system uses a simple genetic algorithm with the strength of the classifiers as the fitness function to select two classifiers to crossover and mutate to create two new and, hopefully, better classifiers. Classifiers with a higher strength have a higher probability of being selected for reproduction.

XCS is a kind of classifier system. There are two major differences between XCS and traditional classifier systems:

As mentioned above, each classifier has a strength parameter that measures how useful the classifier has been in the past. In traditional classifier systems, this strength parameter is commonly referred to as the predicted payoff and is the reward that the classifier expects to receive if its action is executed. The predicted payoff is used to select classifiers to return actions to the environment and also to select classifiers for reproduction.

In XCS, the predicted payoff is also used to select classifiers for returning actions but it is not used to select classifiers for reproduction. To select classifiers for reproduction and for deletion, XCS uses a fitness measure that is based on the accuracy of the classifier's predictions. The advantage to this scheme is that since classifiers can exist in different environmental niches that have different payoff levels and if we just use predicted payoff to select classifiers for reproduction then our population will be dominated by classifiers from the niche with the highest payoff giving an inaccurate mapping of the solution space.

The other difference is that traditional classifier systems run the genetic algorithm on the entire population while XCS uses a niche genetic algorithm. During the course of the XCS algorithm, subsets of classifiers are created. All classifiers in the subsets have conditions that match a given input. The genetic algorithm is run on these smaller subsets. In addition, the classifiers that are selected for mutation are mutated in such a way so that after mutation the condition still matches the input.

Shifting Balance Genetic Algorithm (SBGA)

The SBGA is a module, which can be plugged into a GA, intended to enhance a GA's ability to adapt to a changing environment. A solution that can thrive in a dynamic environment is advantageous.

Cellular Genetic Algorithm (CGA)

The CGA is another attempt at finding an optimal solution in a dynamic environment. A concern of genetic algorithms is that they will find a good solution to a static instance of the problem but will not quickly adapt to a fluctuating environment.

GENETIC PROGRAMMING

Genetic programming (GP) is an extension of genetic algorithms. It is a technique for automatically creating computer programs to solve problems. While GAs search a solution space, GPs search the space of computer programs. New programs can be tested for fitness to achieve a stated objective.

"ANT" ALGORITHMS

An ant algorithm uses a colony of artificial ants, or cooperative agents, designed to solve a particular problem. The ants are contained in a mathematical space where they are allowed to explore, find, and reinforce pathways (solutions) in order to find the optimal ones. Unlike the real-life case, these pathways might contain very complex information. When each ant completes a tour, the pheromones along the ant's path are reinforced according to the fitness (or "goodness") of the solution the ant found. Meanwhile, pheromones are constantly evaporating, so old, stale, poor information leaves the system. The pheromones are a form of collective memory that allows new ants to find good solutions very quickly; when the problem changes, the ants can rapidly adapt to the new problem. The ant algorithm also has the desirable property of being flexible and adaptive to changes in the system. In particular, once learning has occurred on a given problem, ants discover any modifications in the system and find the new optimal solution extremely quickly without needing to start the computations from scratch.

Possible applications to embodiments of the present invention are:

- Search the space of all possible offers to find the offers that are most likely to be accepted
- Search the space of all possible offers to find the most profitable offers that are likely to be accepted
- Evolutionary algorithms can be used together with data mining solutions. For example, a data mining solution could return a score representing the likelihood that an offer will be accepted. Each offer item could have many scores based on different parts of the order. An evolutionary algorithm could be used to devise a strategy for selecting an item based on the collection of scores.

The genetic algorithm XCS and a statistical modeling technique may be combined to score all the offers. An evolutionary strategy known as Explore/Exploit may be used to select offers from the offer pool. Reinforcement learning may be used to improve the system.

The score of an offer should reflect the likelihood that an offer will be accepted given a particular order and may also include the relative value of an offer to an owner. Scores may also include information about how well an offer adheres to other business drivers or metrics such as profitability, gross margin, inventory availability, speed of service, fitness to current marketing campaigns, etc.

For example, in addition to those listed above, an order consists of many parts: the cashier, the register, the destination, the items ordered, the offer price, the time of day, the weather outside, etc. The BioNet divides the pieces of the order into a discrete part and a continuous part. Each part is scored independently and then the scores are combined to reach a final "composite" score for each item.

The discrete part of the order consists of the parts of the order that are disparate attributes: e.g., the cashier, the day of the week, the month, the time of day, the register and the destination. The XCS algorithm is used on the discrete part to arrive at a score.

The continuous part of the order consists of those parts that are not discrete attributes: the ordered items and the offer price. Conditional probabilities are used to score the continuous attributes. Another way to look at the two pieces is as a Variable part and an Invariable part. The variable part consists of the parts of the order that are likely to change from order to order, the items ordered and the offer price, while the invariable part consists of the stuff which is likely to be common among many orders, the cashier, register, etc.

XCS

In order to apply the XCS algorithm, the order is first translated to a bit string of 1's and 0's. Only the so-called discrete parts of the order are translated. The ordered items and offer price are ignored. The population of classifiers is searched for all classifiers that match the order. The action of the classifier represents an offer item. By randomly creating any missing classifiers, the XCS algorithm guarantees that there exists at least one classifier for each possible offer item. The predicted payoffs of the classifiers are averaged to compute a score for each offer item. This score is combined with the score computed by the conditional probabilities to arrive at a final score for each offer item.

CONDITIONAL PROBABILITIES

Naïve Bayes may be used to calculate the conditional probability of an item being accepted given some ordered items and an offer price. Each ordered item and the offer price are treated as independent and equally important pieces of information. The conditional probabilities are calculated using Baye's Rule. Baye's Rule computes the posterior probability of a hypothesis H being true given evidence E:

Baye's Rule: P(H|E) = (P(E|H)P(H)) / P(E)

In our case, the hypothesis is "Item X will be Accepted" and the evidence is the ordered items and the offer price. P(H) is called the "prior probability" or the probability of the Hypothesis in the absence of any evidence.

Since independence was assumed, the probabilities can be multiplied so the actual calculation is as follows:

[Product of for all items in the order[P(item|Offer Accepted)] * P(Offer Accepted) * P(Offer Price | Offer Accepted)] / P(Evidence)

Note that P(Evidence) may be ignored since it disappears as it is normalized.

The probabilities P(E|H) and P(H) are calculated from observed frequencies of occurrences. One facet different from classic data mining problems is that the environment is in a constant state of flux. The parameters that influence the acceptance or decline of an offer may vary from day to day or from month to month. To account for this, in various embodiments of the present invention, the system constantly adapt itself. Instead of using observed frequencies from the beginning of time, the only the most recent transactions are used.

Since the probabilities are multiplied, any P(E|H) or P(H) that is 0 will veto all the other probabilities. In the case of 0 probabilities, the Laplace estimator technique of adding 1 to the numerator and denominator is used.

Once all the offers have been scored, an Explore/Exploit scheme is used to select offers from the offer pool. To select the first item, the system randomly chooses with no bias either Explore or Exploit. If Explore is chosen then caution is thrown to the wind, the scores are ignored and an item is randomly selected from the offer pool. If Exploit is chosen then the item with the best score is selected. So, we use Explore to explore the space of all possible offers and we use Exploit to exploit the knowledge that we have gained. To select the second item, the system again randomly chooses between Explore and Exploit. By employing both Explore and Exploit, the system achieves a nice balance between acquiring knowledge and using knowledge. As a side effect, the Explore strategy also thwarts customer gaming. By periodically throwing in random offers, it is hard to anticipate the system. The problem with exploring is that very bad offers like offering a soda to an order containing a soda can still be presented. To reduce the likelihood but not eliminate the known bad offers, two kinds of Explore, "Completely Random" and "Somewhat Random", are used. Completely Random is as discussed already. Somewhat Random selects an item with an "OK" score.

The system learns by receiving reinforcement from the environment. After an offer is presented, an outcome of either accept, cancel or decline is returned to the system. Both XCS and the observed frequencies of acceptance are updated based on the outcome.

EVOLUTIONARY ALGORITHMS REFERENCES

One of ordinary skill in the art may refer to the following references which describe Evolutionary Algorithms:

Genetic Algorithms, David E. Goldberg 1989 Addison-Wesley

An Introduction to Genetic Algorithms, Melanie Mitchell, 1999, MIT Press

Probabilistic Reasoning in Intelligent Systems, Judea Pearl, 1988, Morgan Kaufmann Publishers, Inc.

An Algorithmic Description of XCS, Martin Butz, Stewart Wilson, IlliGAL Report No. 2000017, April 2000.

Enhancing the GA's Ability to Cope with Dynamic Environments, Mark Wineberg, Franz Oppacher, Proceedings of the Genetic and Evolutionary Computation Conference, July 2000.

An Empirical Investigation of Optimisation in Dynamic Environments Using the Cellular Genetic Algorithm, Michael Kirley, David G. Green, Proceedings of the Genetic and Evolutionary Computation Conference, July 2000.

Genetic Programming (Complex Adaptive Systems), John Koza, 1992, MIT Press

Statistical or Traditional Self-improving Method

Standard statistical modeling methods can be used to achieve similar results of GA or other algorithms.

PROFIT ENGINE CALCULATIONS

In order to maximize the return on Digital Deal offers, a method could be implemented to make the most profitable offers to the customer with the highest probability of acceptance. One way to accomplish this would be to add a new offer property: Popularity. If we weight the popularity of an offer high and the profitability high, we maximize the return.

Testing has shown that the likelihood of an item being accepted is influenced greatly by the cost of the order. In order to calculate the popularity of an offer item we regard the offer item with respect to the cost of the entire order and the previous acceptance rate of that item. Note: This approach will be extended to handle the issue of popularity based on other factors such as the total discount or value proposition.

Calculating the Popularity:

In order to calculate the popularity we define a function that returns the popularity of a given menu item based on the order total. The popularity is the predicted likelihood of acceptance at a given order total.

The popularity function is a least squares curve fit to the historical acceptance rates of an item. A second degree polynomial is being used for the curve fit. The popularity function is defined as follows:

Popularity = a x^2 + b x + c
Where
X = Order total
A, B, C = popularity coefficients

Determining the data set to use for the curve fit is done as follows. The range of offers are divided up into increments (e.g. 50¢). All of the offers within a given range are averaged and the average take rate per increment is set. A curve is fit through the average take rate samples and the coefficients for the above function are calculated. These coefficients are stored in the database for each menu item.

A program may be run at a predetermined time (e.g. End of Day) to calculate the Popularity coefficients for each menu item. The user will need to set the order total increment and the minimum number of points per increment. This will allow for tuning of the system.

HANDLING LIMITATIONS

In order to allow for increments that don't have sufficient data the following technique will be used. If an increment range (e.g. 0¢ – 25¢) has less than the minimum number of points it is merged with the next increment. This continues until the minimum number of points are found in an increment. If there is insufficient data to fit a curve (3 valid intervals) then a linear function (2 valid intervals) or a constant (1 or less intervals) will be used.

VERIFICATION OF A VALID CURVE FIT

Each curve can be checked to see if there is a valid trend (meets a given threshold for standard deviation). If the curve fit is determined to be invalid then the average take rate for all offers of this item will be used as the popularity function.

PUTTING IT ALL TOGETHER

The goal of implementing the popularity attribute per offer is to score the offers according to the predicted probability of acceptance. The scoring engine will provide a method for weighting the popularity of an item in relation to the other score parameters. So, in order to maximize the most profitable offers and those most likely to be accepted you would weight the popularity and the profitability higher than any other score parameters.

References

One of ordinary skill in the art may refer to the following references for a description of learning systems.

[1]. MITCHELL TM. *MACHINE LEARNING*. 1997. MCGRAW-HILL: BOSTON

[2]. KAELBLING LP, LITTMANN ML, MOORE AW. 1996. REINFORCEMENT LEARNING: A SURVEY. *J. ARTIFICIAL INTELLIGENCE RESEARCH* 4: 237-285

[3]. Crites RH, and Barto AG. Improving Elevator Performance Using Reinforcement Learning. *Advances in Neural Information Processing Systems 8*. MIT: Cambridge.

[4]. Kaelbling LP. Associative Reinforcement Learning: A Generate and Test Algorithm. Kluwer: Boston.

[5]. Anderson CW. Approximating a Policy Can be Easier Than Approximating a Value Function. 2000. *Colorado State University Technical Report: CS-00-01*

[6]. Kaelbling LP. Associative Reinforcement Learning: Functions in *k*-DNF. Kluwer: Boston.

[7]. Opitz D, Maclin R. 1999. Popular Ensemble Methods: An Empirical Study. *J. Artifical Intelligence Research*. **11**: 169-198.

[8]. Opitz D, Shavlik JW. 1997. Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies. *J. Artificial Intelligence Research* **6**: 177-209.

[9]. Kachigan SK. 1991. *Multivariate Statistical Analysis*. Radius Press: New York.

[10]. Koza J. Genetic Programming III.

[11]. Gerhart JC, Kirschner MW. 1997. Cells, Embros and Evolution. Blackwell Sciences.